

14. Programmieren mit Löchern

Die Basismittel für Frameworks

Prof. Dr. rer. nat. Uwe Aßmann
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
Version 19-0.6, 12.04.19

- 1) Abstrakte Klassen und Schnittstellen
- 2) Generische Klassen



DRESDEN
concept
Exzellenz aus
Wissenschaft
und Kultur

Begleitende Literatur

2

Softwaretechnologie (ST)

- ▶ Das **Vorlesungsbuch** von Pearson: **Softwaretechnologie für Einsteiger**. Vorlesungsunterlage für die Veranstaltungen an der TU Dresden. Pearson Studium, 2014. Enthält ausgewählte Kapitel aus:
 - UML: Harald Störrle. UML für Studenten. Pearson 2005. Kompakte Einführung in UML 2.0.
 - Softwaretechnologie allgemein: W. Zuser, T. Grechenig, M. Köhle. Software Engineering mit UML und dem Unified Process. Pearson.
 - Bernd Brügge, Alan H. Dutoit. Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java. Pearson Studium/Prentice Hall.
 - Erhältlich in SLUB
- Noch ein sehr gutes, umfassend mit Beispielen ausgestattetes Java-Buch:
 - C. Heinisch, F. Müller, J. Goll. Java als erste Programmiersprache. Von Einsteiger zum Profi. Teubner.
- ▶ Für alle, die sich nicht durch Englisch abschrecken lassen:
- ▶ Safari Books, von unserer Bibliothek SLUB gemietet:
 - <http://proquest.tech.safaribooksonline.de/>
- ▶ Free Books: <http://it-ebooks.info/>
 - Kathy Sierra, Bert Bates: Head-First Java <http://it-ebooks.info/book/255/>



Obligatorische Literatur

3

Softwaretechnologie (ST)

- ▶ ST für Einsteiger Kap. 9, Teil II (Störrle, Kap. 5.2.6, 5.6)
- ▶ Zuser Kap 7, Anhang A
- ▶ Java
 - <http://docs.oracle.com/javase/tutorial/java/index.html> is the official Oracle tutorial on Java classes
 - Balzert LE 9-10
 - Boles Kap. 7, 9, 11, 12

"Objektorientierte Softwareentwicklung"
Hörsaalübung
Fr, 13:00 HSZ 03, Dr. Demuth

Ziele

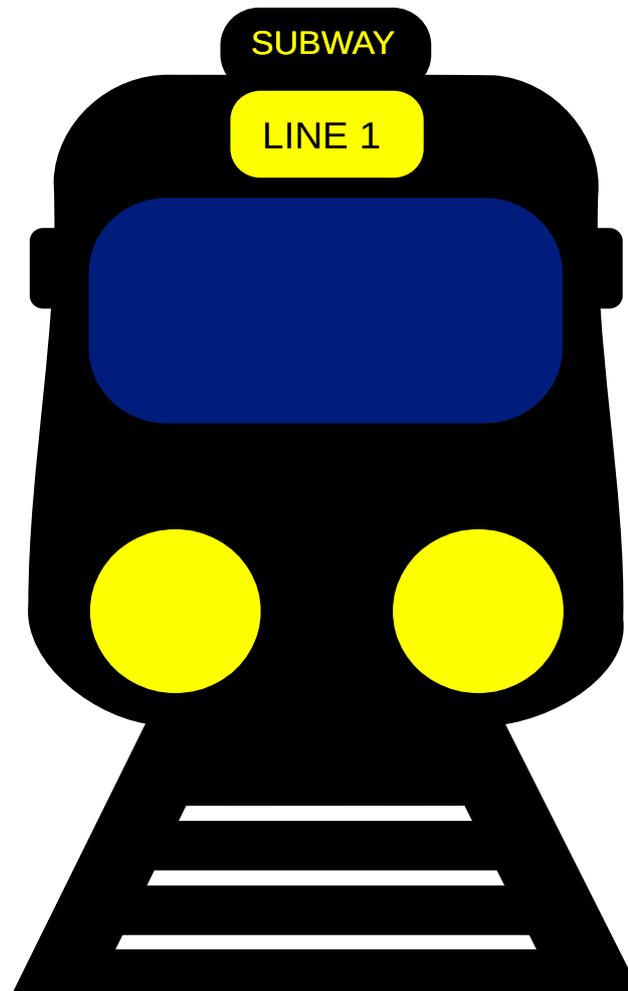
4 Softwaretechnologie (ST)

- ▶ Abstrakte Klassen und Schnittstellen verstehen
- ▶ Generische Typen zur Vermeidung von Fehlern (Nachbartypschränken)

Java Herunterladen

5 Softwaretechnologie (ST)

- ▶ Das Java Development Kit (JDK) 12
- ▶ <http://openjdk.java.net/>



Frameworks – Komponenten mit Löchern

Def.: Ein Framework bildet eine besondere Art von wiederverwendbarer Komponente, denn es ist mit Löchern programmiert, an denen man es erweitern kann.

- ▶ Ein Framework hat immer einen Erweiterungs-Ring (die Löcher).
- ▶ Die Löcher machen Vorgaben für Erweiterungen, z.B. geben sie Typen oder Typschränken für Erweiterungen vor
- ▶ Damit kann der Framework-Konstrukteur dem Anwendungs-Konstrukteur Vorgaben machen, um die Funktionsfähigkeit des Frameworks in der Anwendung zu garantieren
- ▶ Komponenten und Frameworks bilden die Basis aller objekt-orientierter Wiederverwendung in Firmen.

14.1 Schnittstellen und Abstrakte Klassen für das Programmieren von Löchern

Typen können verschiedene Formen annehmen. Eine partiell spezifizierte Klasse (Schnittstelle, abstrakte Klasse, generische Klasse) macht Vorgaben für Anwendungsentwickler

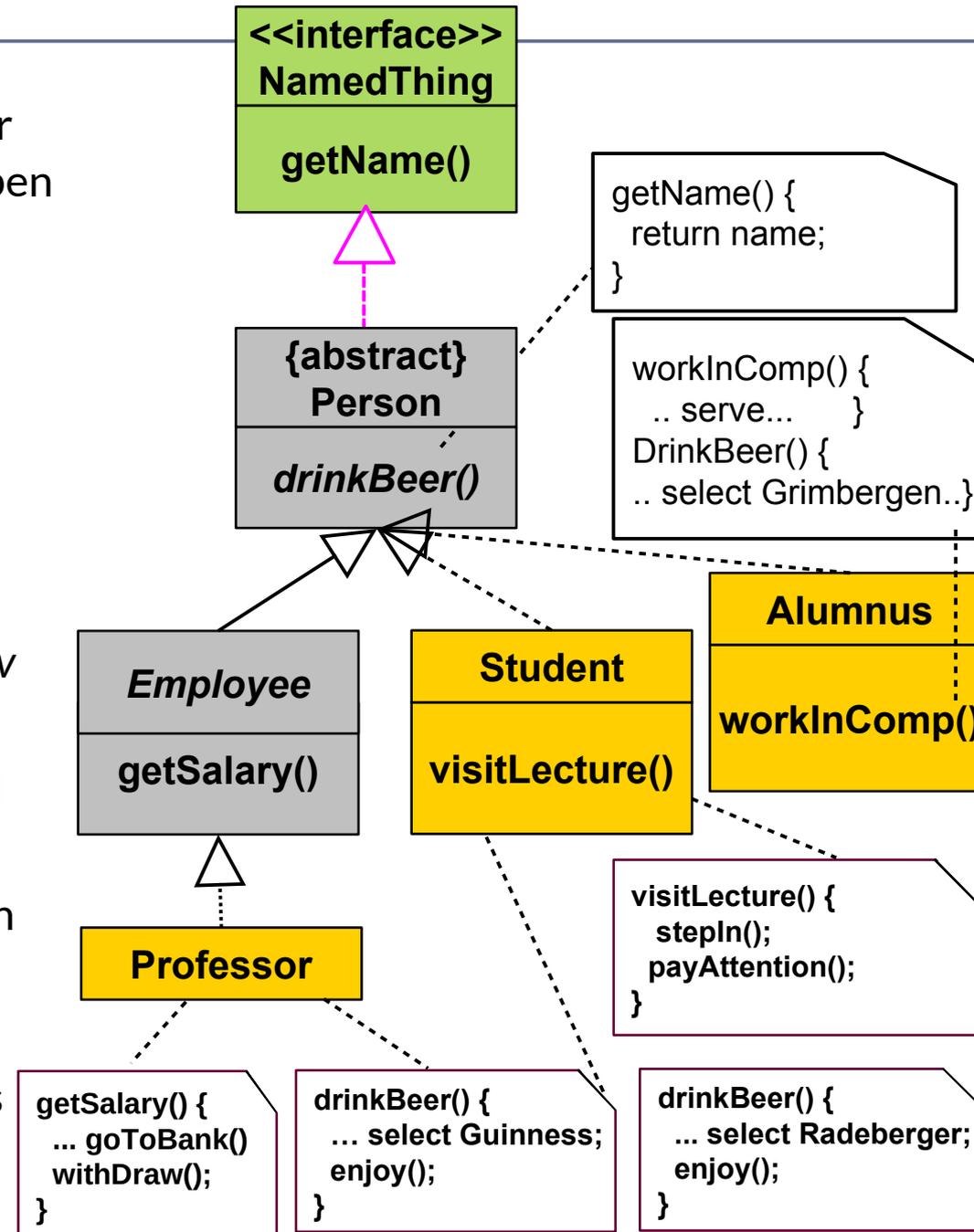


Schnittstellen und Abstrakte Klassen bilden “Haken”, an die man Klassenimplementierungen anhängen kann

8

Softwaretechnologie (ST)

- ▶ Beim Entwurf von Bibliotheken soll für Anwendungen eine Struktur vorgegeben werden, an die sich alle Anwendungsprogrammierer halten müssen
- ▶ **Vorsehen von “Haken” (hooks)** in der Vererbungshierarchie:
- ▶ **Abstrakte Klassen** werden mit einem speziellen Markierer (tagged value) gekennzeichnet ({abstract}) oder *kursiv* gemalt
- ▶ **Schnittstellen** beschreiben einen Teil der Funktionalität eines Objekts
 - In UML werden sog. Stereotypen vergeben, um Schnittstellen zu kennzeichnen (<<interface>>)
 - Sie dienen dazu, Verhalten eines Objektes in einem bestimmten Kontext festzulegen



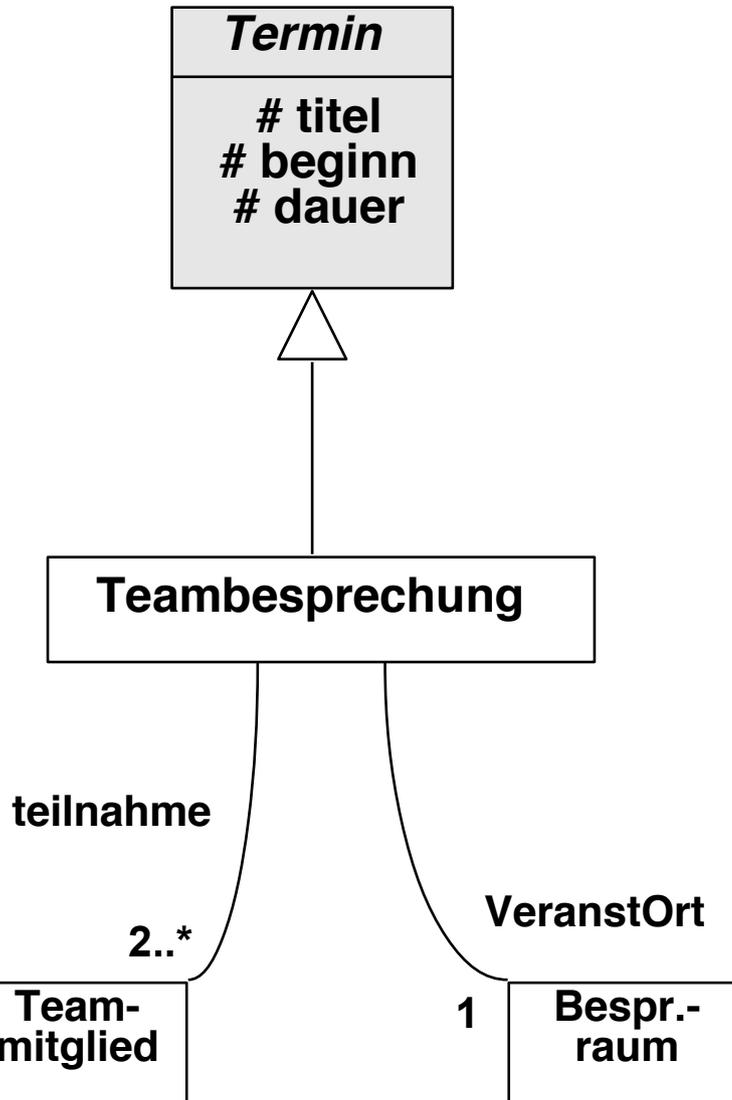
Schnittstellen und Klassen in Java geben “Hooks” vor (“abstract”)

```
interface NamedThing {
    String getName(); // no implementation
}
abstract class Person implements NamedThing {
    String name;
    String getName() { return name; } // implementation exists
    abstract void drinkBeer(); // no implementation
}
abstract class Employee extends Person {
    abstract void getSalary(); // no implementation
}
class Professor extends Employee { // concrete class
    void getSalary() { goToBank(); withdraw(); }
    void drinkBeer() { .. select Guinness(); enjoy(); }
}
class Student extends Person { // concrete class
    void visitLecture() { stepIn(); payAttention(); }
    void drinkBeer() { .. select Radeberger(); enjoy(); }
}
```

Schnittstellen und Klassen in Java geben “Hooks” vor

```
interface NamedThing {
    String getName(); // no implementation
}
abstract class Person implements NamedThing {
    String name;
    String getName() { return name; } // implementation exists
    abstract void drinkBeer(); // no implementation
}
abstract class Employee extends Person {
    abstract void getSalary(); // no implementation
}
class Professor extends Employee { // concrete class
    void getSalary() { goToBank(); withdraw(); }
    void drinkBeer() { .. select Guinness(); enjoy(); }
}
class Student extends Person { // concrete class
    void visitLecture() { stepIn(); payAttention(); }
    void drinkBeer() { .. select Radeberger(); enjoy(); }
}
class Alumnus extends Person {
    // new concrete class must fit to Person and NamedThing
    void workInComp() { .. serve... }
    void drinkBeer() { ...select Wine... }
}
```

Laufendes Beispiel Terminverwaltung



```
abstract class Termin {
    ...
    protected String titel;
    protected Hour beginn;
    protected int dauer;
    ...
};
```

```
class Teambesprechung
    extends Termin {
    private Teammitglied[] teilnahme;
    private BesprRaum veranstOrt;
    ...
};
```



Beispiel (2): Abstrakte Klassen und Abstrakte Operationen

12 Softwaretechnologie (ST)

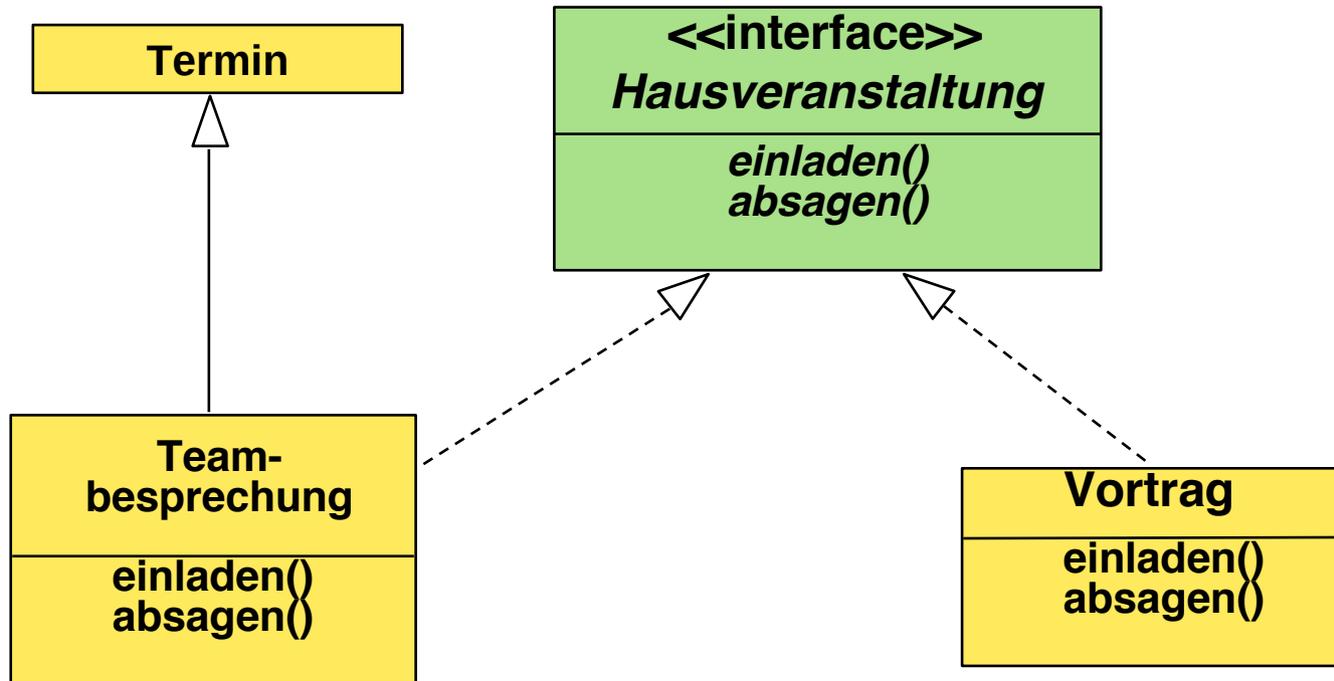
```
abstract class Termin {  
    ...  
    protected String titel;  
    protected Hour beginn;  
    protected int dauer;  
    ...  
    abstract boolean verschieben (Hour neu);  
};
```

Jede abstrakt deklarierte Methode muß in einer Unterklasse realisiert werden - sonst können keine Objekte der Unterklasse erzeugt werden!

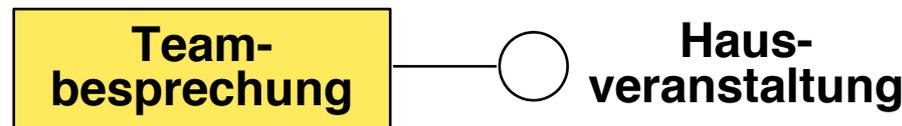
```
class Teambesprechung  
    extends Termin {  
    ...  
    boolean verschieben (Hour neu) {  
        boolean ok =  
            abstimmen(neu, dauer);  
        if (ok) {  
            beginn = neu;  
            raumFestlegen();  
        };  
        return ok;  
    };  
};
```

```
class Betriebsversammlung  
    extends Termin {  
    ...  
    boolean verschieben (Hour neu) {  
        beginn = neu;  
        raumFestlegen();  
        return ok;  
    };  
};
```

Einfache Vererbung von Typen durch Schnittstellen



Hinweis: “Lutscher”-Notation (*lollipop*) für Schnittstellen „Klasse bietet Schnittstelle an“:



Vergleich von Schnittstellen und abstrakte Klassen

Abstrakte Klasse

**Enthält Attribute
und Operationen**

**Kann Default-Verhalten
festlegen**

**Wiederverwendung von
Schnittstellen und Code,
aber keine Instanzbildung**

**Default-Verhalten kann in
Unterklassen überdefiniert
werden**

**Java: Unterklasse kann nur
von einer Klasse erben**

Schnittstelle (voll abstrakt)

**Enthält nur Operationen
(und ggf. Konstante)**

**Kann kein Default-Verhalten
festlegen**

Redefinition unsinnig

**Java und UML: Eine Klasse kann
mehrere Schnittstellen
implementieren**

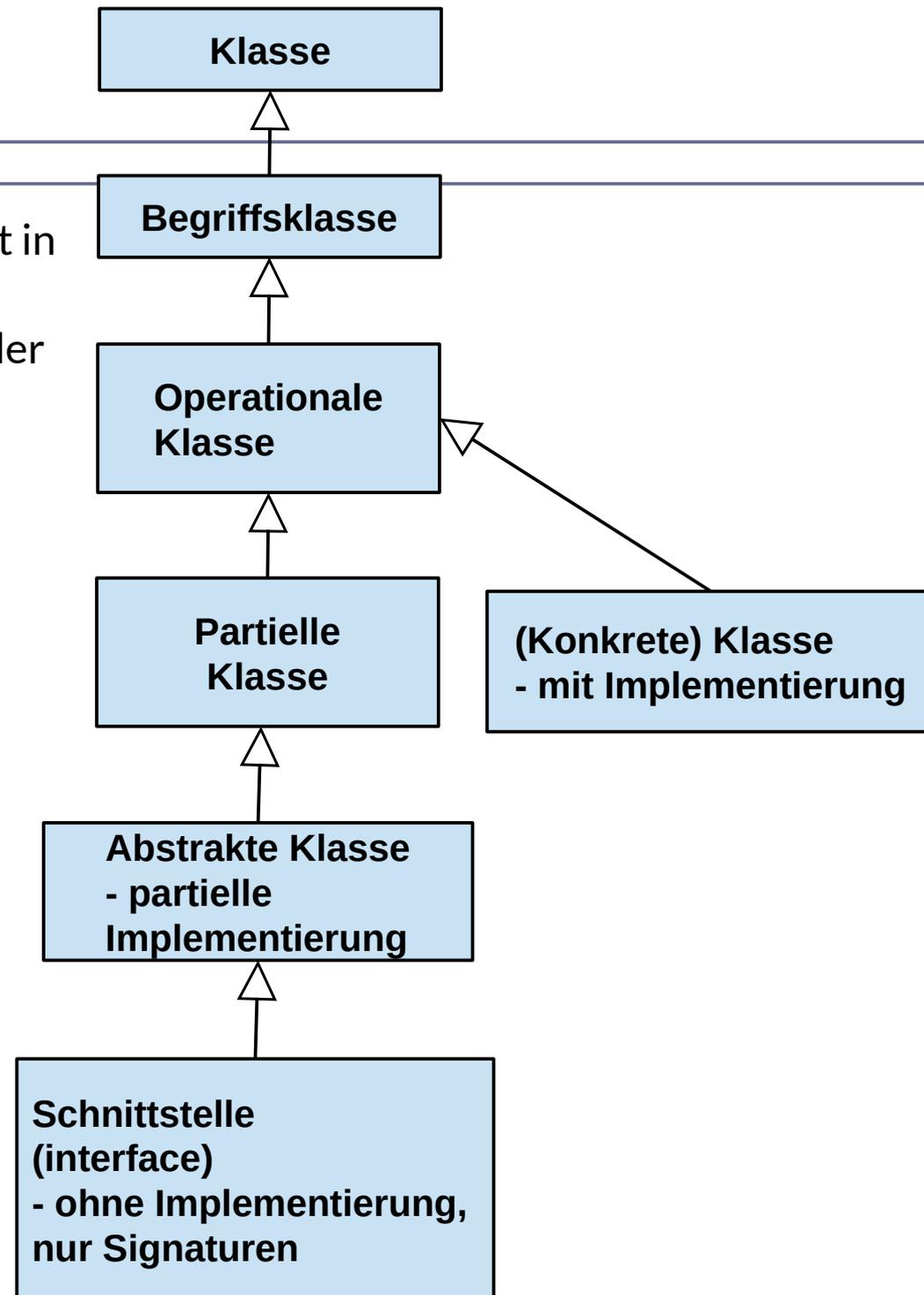
**Schnittstelle ist eine spezielle
Sicht auf eine Klasse**

Q2: Begriffshierarchie von Klassen (Erw.)

15

Softwaretechnologie (ST)

- ▶ *Operationale Klassen* werden unterteilt in Klassen mit, ohne, und mit Implementierung einer Untermenge der Operationen
- ▶ *Schnittstellen* und *Abstrakte Klassen* dienen dem Teilen von Typen und partiellem Klassen-Code



14.2. Generische Klassen (Klassenschablonen, Template-Klassen, Parametrische Klassen)

... bieten eine weitere Art, mit Löchern zu programmieren, um Vorgaben zu machen

- ▶ Generische Klassen lassen den Typ von einigen Attributen und Referenzen offen (“generisch”)
- ▶ Sie ermöglichen typisierte Wiederverwendung von Code (Ausfaktorisierung von Gemeinsamkeiten)
- ▶ Sie helfen, Nachbarn zu spezialisieren



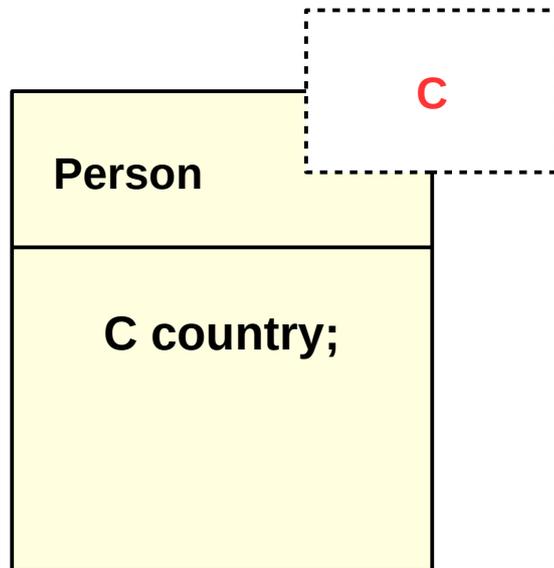
Generische Klassen

17

Softwaretechnologie (ST)

Def.: Eine **generische (parametrische, Template-) Klasse** ist eine Klassenschablone, die mit einem oder mehreren Typparametern (für Attribute, Referenzen, Methodenparameter) versehen ist.

▶ In UML



▶ In Java

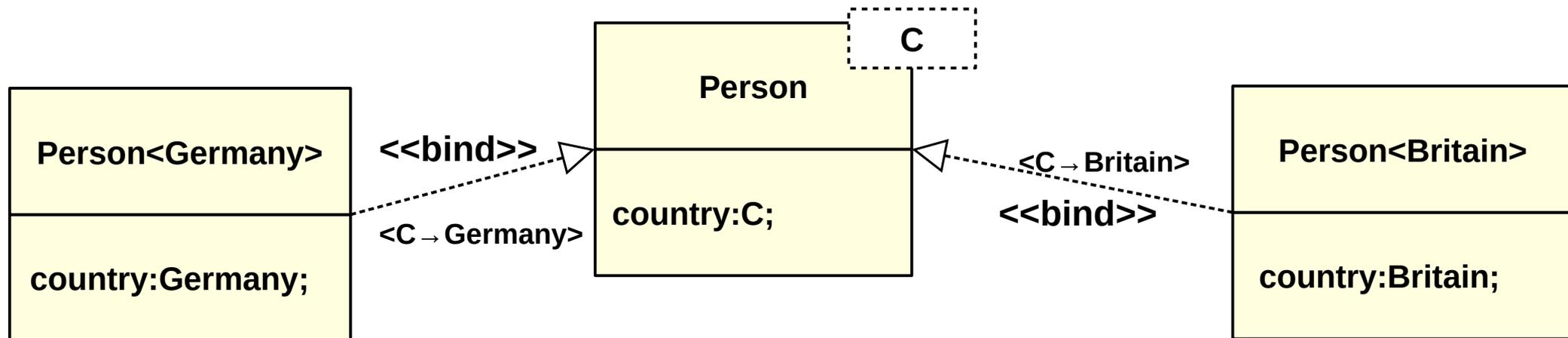
– Sprachregelung: “Person of C”

```
// Definition of a generic class
class Person<C> {
    C country;
}
```

```
/* Type definition using a generic type */
Person<Germany> egon;
Person<Britain> john;
```

Feinere statische Typüberprüfung für Attribute

- ▶ Zwei Attributtypen, die durch Parameterisierung aus einer generischen Klassenschablone entstanden sind, sind nicht miteinander kompatibel
- ▶ Der Übersetzer entdeckt den Fehler (**statische Typprüfung**)
- ▶ Die generische Klasse beschreibt das Gemeinsame (Generalisierung); der Parameter die Verschiedenheiten; die ausgeprägte Klasse die Spezialisierung



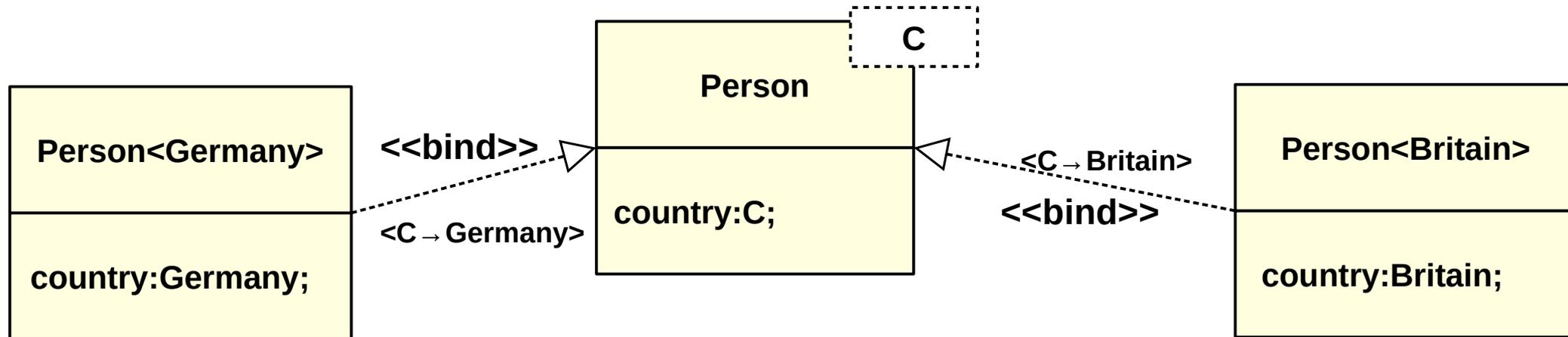
```
/* Type definition and initialization with object */
Person<Germany> egon = new Person<Germany>;
Person<Britain> john = new Person<Britain>;

/* Checks of assignments can use the improved typing */
john = egon;
```



Einsatzzweck: Typen von Nachbarn vorgeben

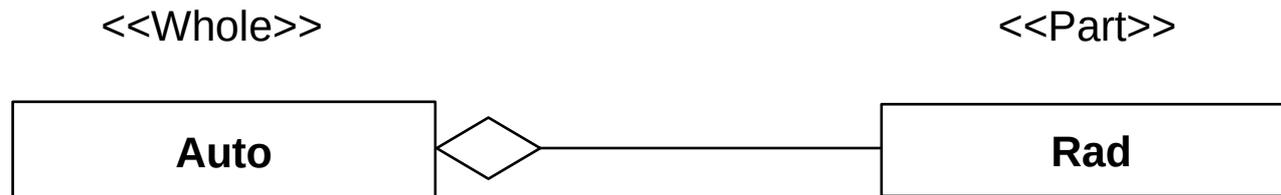
- ▶ **Def.:** Wenn ein formaler Typparameter einer generischen Klasse einen Nachbarn beschreibt, nennen wir ihn **Nachbartypschränke**



- ▶ **Bsp.:** C gibt für die generische Klasse den Typ des Landes country vor (Nachbar)
- ▶ C ist eine Nachbartypschränke

Einsatzzweck: Typen von Nachbarn vorgeben, Typsichere Aggregation (has-a)

- ▶ **Def.:** Wenn eine Assoziation den Namen „hat-ein“ oder „besteht-aus“ tragen könnte, handelt es sich um eine **Aggregation** zwischen einem *Aggregat*, dem *Ganzen*, und seinen *Teilen* (Ganzes/Teile-Relation, whole-part relationship).
 - Die auftretenden Aggregationen bilden auf den Objekten immer eine transitive, antisymmetrische Relation (einen gerichteten zyklensfreien Graphen, *dag*).
 - Ein Teil kann zu mehreren Ganzen gehören (*shared*), zu einem Ganzen (*owns-a*) und exklusiv zu einem Ganzen (*exclusively-owns-a*)



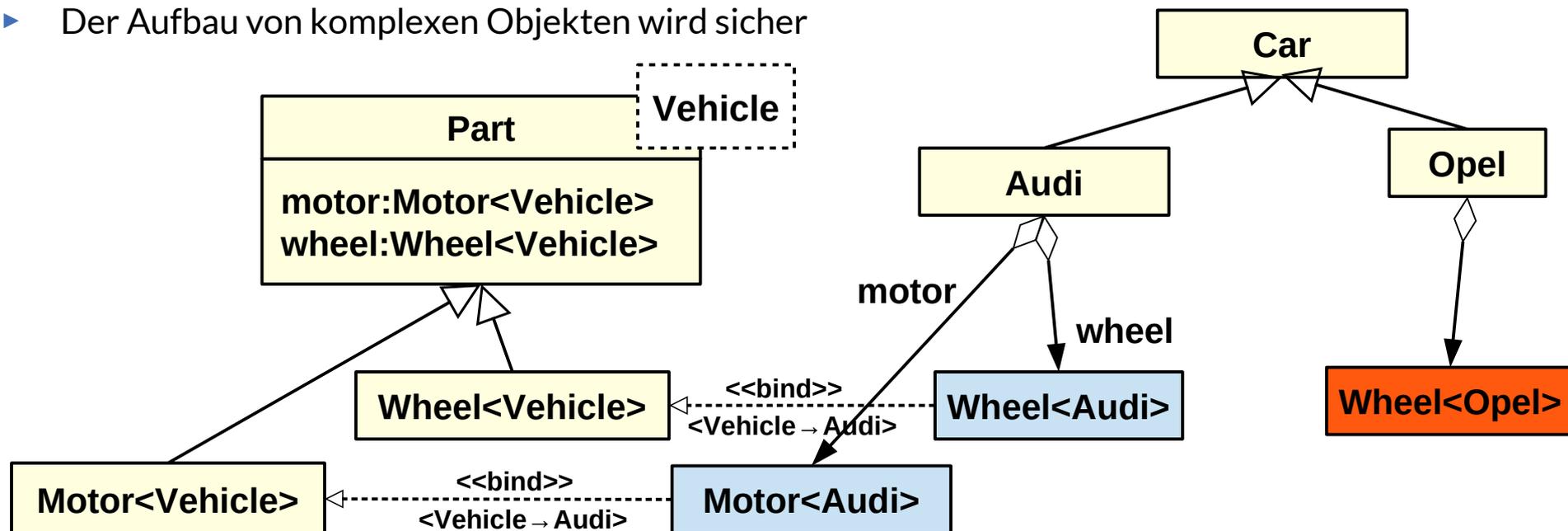
Lies: „Auto *hat ein* Rad“

Einsatzzweck: Typen von Nachbarn vorgeben, Typsichere Aggregation (has-a)

21

Softwaretechnologie (ST)

- ▶ Generische Klassen können Typen für Ganz-Teile-Beziehungen vorgeben
- ▶ Der Aufbau von komplexen Objekten wird sicher



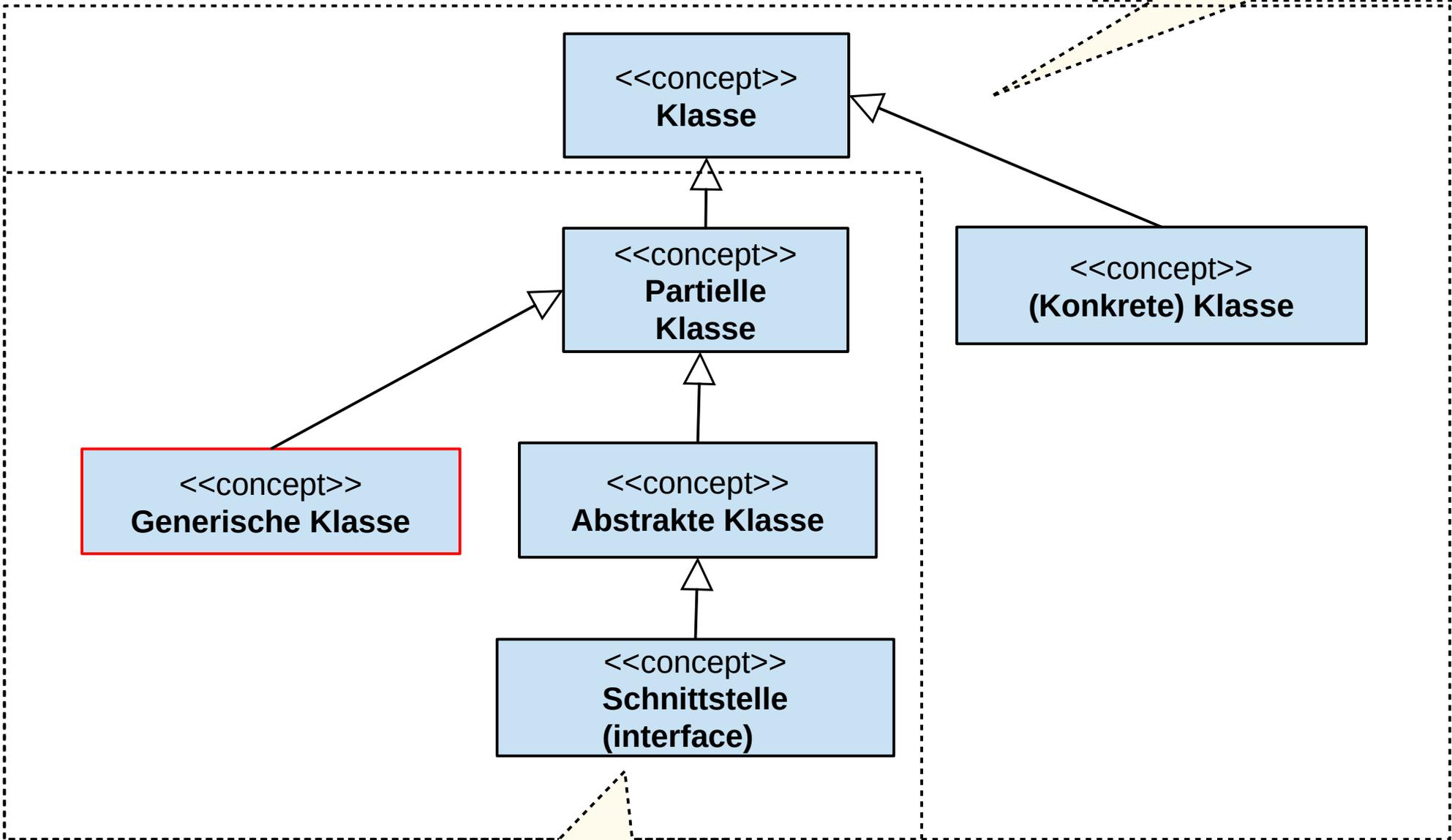
```
/* Type definition and initialization with object */
Motor<Audi> motorOfAudi = new Motor<Audi>;
Wheel<Audi> wheelOfAudi = new Wheel<Audi>;
Wheel<Opel> wheelOfOpel = new Wheel<Opel>;

/* Checks of assignments can use the improved typing */
audi = new Audi();
audi.motor = motorOfAudi;
audi.wheel = wheelOfOpel;
```



Q2: Begriffshierarchie von Klassen (Erweiterung)

Fürs Programmieren mit
Gemeinsamkeiten



Fürs Programmieren mit
Löchern

- ▶ Schnittstellen als auch abstrakte Klassen erlauben es, Anwendungsprogrammierern Struktur vorzugeben
 - Sie definieren “Haken”, in die Unterklassen konkrete Implementierungen schieben
 - Schnittstellen sind vollständig abstrakte Klassen
- ▶ Generische Klassen ermöglichen typsichere Wiederverwendung von Code über Typ-Parameter → der Compiler meldet mehr Fehler

Warum ist das wichtig?

- ▶ **Bau von Frameworks (Rahmenwerken)** ist eines der Hauptprobleme des Software Engineering
 - Von Projekt zu Projekt
 - Von Produkt zu Produkt (Produktfamilien, Produktlinien)
- ▶ Abstrakte Klassen, Schnittstellen und generische Klassen können Code-Replikate und Code-Explosion weitgehend vermeiden und **gleichzeitig Vorgaben für Erweiterungen machen**
- ▶ Wiederverwendung **mit Frameworks** ist das Hauptmittel der Softwarefirmen, um **profitabel** arbeiten zu können

Verständnisfragen

- ▶ Geben Sie eine Begriffshierarchie des Klassenbegriffs an. Welche Klassenarten kennen Sie? Wie spezialisieren sie sich?
- ▶ Erklären Sie den Unterschied der “Löcher” in abstrakten Klassen und in generischen Klassen.
- ▶ Warum wird die Wiederverwendung von Software durch Frameworks vereinfacht? Wozu gibt der Framework-Konstrukteur Vorgaben für die Löcher vor?

14. Programmieren mit Löchern

Die Basismittel für Frameworks

Prof. Dr. rer. nat. Uwe Aßmann

Lehrstuhl Softwaretechnologie

Fakultät für Informatik

TU Dresden

Version 19-0.6, 12.04.19

1) Abstrakte Klassen und Schnittstellen

2) Generische Klassen



DRESDEN
Concept
Exzellenz aus
Wissenschaft
und Kultur

Keep your code clean, take your stance:

Use single inheritance.

Instead copy, move it up

Superclass will be its hub.

Breathe into your objects life -

Let them polymorphy drive.

Abstract classes, interface

Shape their childrens' pretty face.

Begleitende Literatur

- ▶ Das **Vorlesungsbuch** von Pearson: **Softwaretechnologie für Einsteiger**. Vorlesungsunterlage für die Veranstaltungen an der TU Dresden. Pearson Studium, 2014. Enthält ausgewählte Kapitel aus:
 - UML: Harald Störrle. UML für Studenten. Pearson 2005. Kompakte Einführung in UML 2.0.
 - Softwaretechnologie allgemein: W. Zuser, T. Grechenig, M. Köhle. Software Engineering mit UML und dem Unified Process. Pearson.
 - Bernd Brügge, Alan H. Dutoit. Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java. Pearson Studium/Prentice Hall.
 - Erhältlich in SLUB
- Noch ein sehr gutes, umfassend mit Beispielen ausgestattetes Java-Buch:
 - C. Heinisch, F. Müller, J. Goll. Java als erste Programmiersprache. Von Einsteiger zum Profi. Teubner.
- ▶ Für alle, die sich nicht durch Englisch abschrecken lassen:
- ▶ Safari Books, von unserer Bibliothek SLUB gemietet:
 - <http://proquest.tech.safaribooksonline.de/>
- ▶ Free Books: <http://it-ebooks.info/>
 - Kathy Sierra, Bert Bates: Head-First Java <http://it-ebooks.info/book/255/>



Safari Tech Books Online verfügbar.

- Katalog der SLUB (www.slub-dresden.de) verzeichnet ca. 30.000 elektronische Bücher mit Bezug zur Informatik, die online gelesen werden können
- <http://proquest.tech.safaribooksonline.de/>

Obligatorische Literatur

- ▶ ST für Einsteiger Kap. 9, Teil II (Störrle, Kap. 5.2.6, 5.6)
- ▶ Zuser Kap 7, Anhang A
- ▶ Java
 - <http://docs.oracle.com/javase/tutorial/java/index.html> is the official Oracle tutorial on Java classes
 - Balzert LE 9-10
 - Boles Kap. 7, 9, 11, 12

"Objektorientierte Softwareentwicklung"
Hörsaalübung
Fr, 13:00 HSZ 03, Dr. Demuth

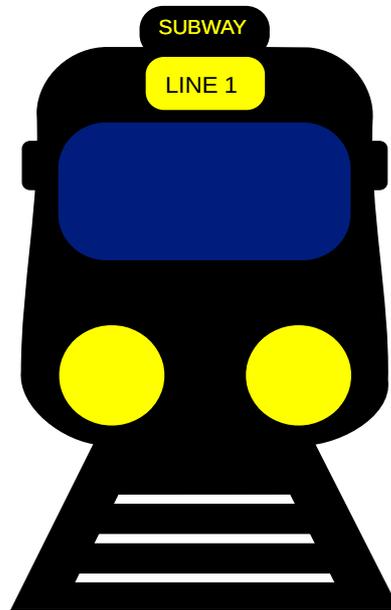
Ziele

- ▶ Abstrakte Klassen und Schnittstellen verstehen
- ▶ Generische Typen zur Vermeidung von Fehlern (Nachbartypschränken)

Java Herunterladen

5 Softwaretechnologie (ST)

- ▶ Das Java Development Kit (JDK) 12
- ▶ <http://openjdk.java.net/>



If you have not yet downloaded Java, and started the compiler and the VM, you are already rather late and in danger to miss the train.

Def.: Ein Framework bildet eine besondere Art von wiederverwendbarer Komponente, denn es ist mit Löchern programmiert, an denen man es erweitern kann.

- ▶ Ein Framework hat immer einen Erweiterungs-Ring (die Löcher).
- ▶ Die Löcher machen Vorgaben für Erweiterungen, z.B. geben sie Typen oder Typschranken für Erweiterungen vor
- ▶ Damit kann der Framework-Konstrukteur dem Anwendungs-Konstrukteur Vorgaben machen, um die Funktionsfähigkeit des Frameworks in der Anwendung zu garantieren
- ▶ Komponenten und Frameworks bilden die Basis aller objekt-orientierter Wiederverwendung in Firmen.

14.1 Schnittstellen und Abstrakte Klassen für das Programmieren von Löchern

Typen können verschiedene Formen annehmen. Eine partiell spezifizierte Klasse (Schnittstelle, abstrakte Klasse, generische Klasse) macht Vorgaben für Anwendungsentwickler



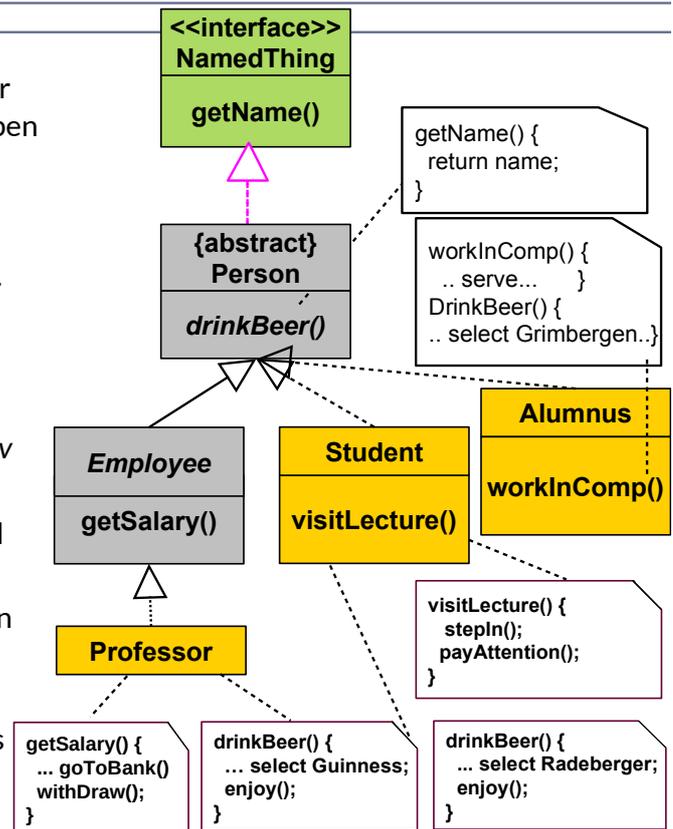
DRESDEN
Concept
Exzellenz aus
Wissenschaft
und Kultur

Schnittstelle:
Loch an Loch – und sie hält doch.

Schnittstellen und Abstrakte Klassen bilden "Haken", an die man Klassenimplementierungen anhängen kann

8 Softwaretechnologie (ST)

- ▶ Beim Entwurf von Bibliotheken soll für Anwendungen eine Struktur vorgegeben werden, an die sich alle Anwendungsprogrammierer halten müssen
- ▶ **Vorsehen von "Haken" (hooks)** in der Vererbungshierarchie:
- ▶ **Abstrakte Klassen** werden mit einem speziellen Markierer (tagged value) gekennzeichnet (`{abstract}`) oder kursiv gemalt
- ▶ **Schnittstellen** beschreiben einen Teil der Funktionalität eines Objekts
 - In UML werden sog. Stereotypen vergeben, um Schnittstellen zu kennzeichnen (`<<interface>>`)
 - Sie dienen dazu, Verhalten eines Objektes in einem bestimmten Kontext festzulegen



In Java programmiert man mit Schablonen:

- Schnittstellen geben Methodensignaturen vor, die von Unterklassen implementiert werden müssen
- Abstrakte Klassen ebenfalls
- Generische Klassen geben Typen für Attribute und Methodenparameter vor, die zu verwenden sind.

Damit schafft es ein Chef-Entwickler, anderen Entwicklern Vorgaben zu machen:

- Der Programmierer einer Bibliothek kann vorgeben, wie man sie benutzt
- Die Entwickler des "Java Development Kits (JDK)" geben uns allen vor, wie wir Anwendungen zu schreiben haben ("Frameworks", Rahmenwerke)

Das ist in nicht-objektorientierten Sprachen bei Weitem nicht so gut möglich. Daher unterstützen OO-Programme Wiederverwendung.

Schnittstellen und Klassen in Java geben “Hooks” vor (“abstract”)

9

Softwaretechnologie (ST)

```
interface NamedThing {
    String getName(); // no implementation
}
abstract class Person implements NamedThing {
    String name;
    String getName() { return name; } // implementation exists
    abstract void drinkBeer(); // no implementation
}
abstract class Employee extends Person {
    abstract void getSalary(); // no implementation
}
class Professor extends Employee { // concrete class
    void getSalary() { goToBank(); withDraw(); }
    void drinkBeer() { .. select Guinness(); enjoy(); }
}
class Student extends Person { // concrete class
    void visitLecture() { stepIn(); payAttention(); }
    void drinkBeer() { .. select Radeberger(); enjoy(); }
}
```

Alle in einer Schnittstelle definierten Methodensignaturen bilden “Löcher”, “Haken”, “hooks”.

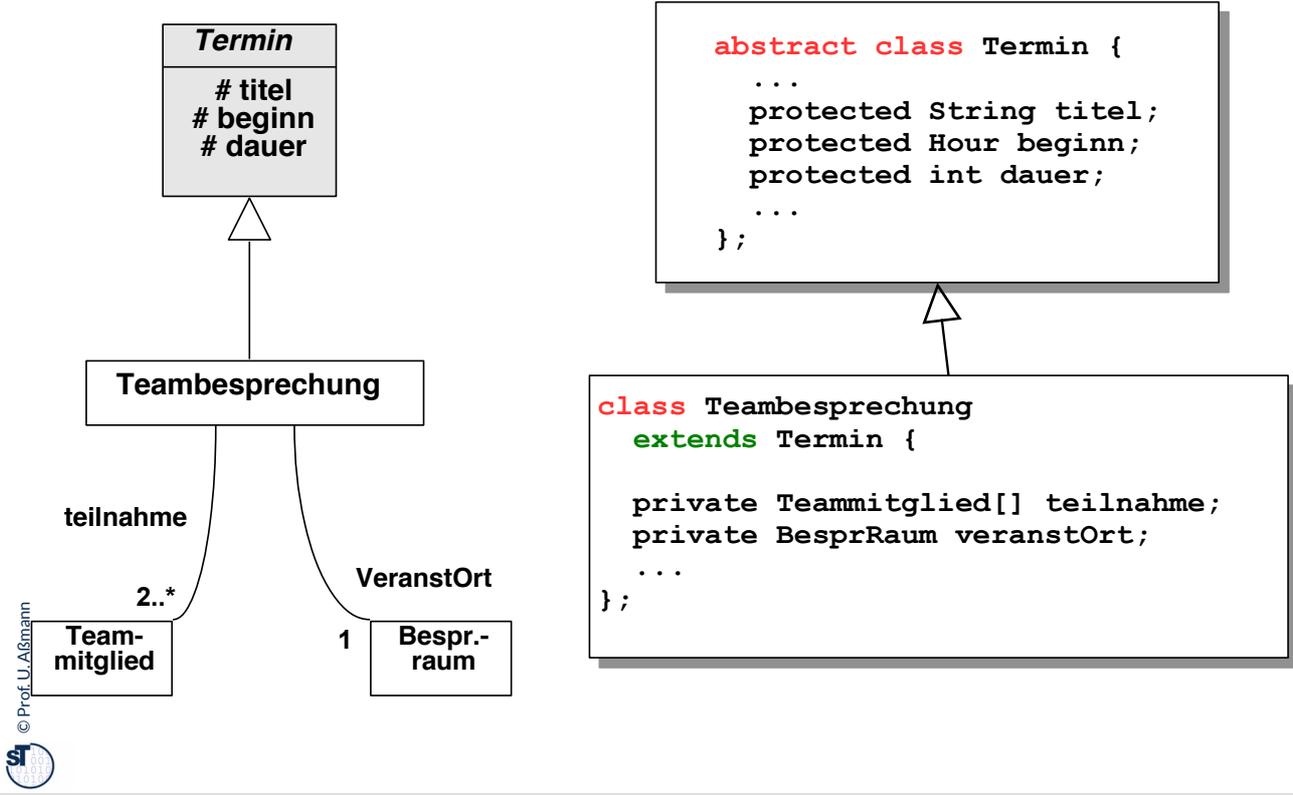
Eine konkrete Unterklasse muss sie alle füllen, also “Methoden einhängen”.

Schnittstellen und Klassen in Java geben "Hooks" vor

```
interface NamedThing {
    String getName(); // no implementation
}
abstract class Person implements NamedThing {
    String name;
    String getName() { return name; } // implementation exists
    abstract void drinkBeer(); // no implementation
}
abstract class Employee extends Person {
    abstract void getSalary(); // no implementation
}
class Professor extends Employee { // concrete class
    void getSalary() { goToBank(); withdraw(); }
    void drinkBeer() { .. select Guinness(); enjoy(); }
}
class Student extends Person { // concrete class
    void visitLecture() { stepIn(); payAttention(); }
    void drinkBeer() { .. select Radeberger(); enjoy(); }
}
class Alumnus extends Person {
    // new concrete class must fit to Person and NamedThing
    void workInComp() { .. serve... }
    void drinkBeer() { ...select Wine... }
}
```

Unterklassen dürfen neue Methoden definieren. Wenn sie aber eine Methodensignatur einer Schnittstelle füllen wollen, müssen sie dieser genau entsprechen.

Laufendes Beispiel Terminverwaltung



Die Klasse Termin definiert keine Methodensignaturen, aber geschützte Attribute. Auch hier können Unterklassen weitere Attribute hinzufügen.

Beispiel (2): Abstrakte Klassen und Abstrakte Operationen

12 Softwaretechnologie (ST)

```
abstract class Termin {  
    ...  
    protected String titel;  
    protected Hour beginn;  
    protected int dauer;  
    ...  
    abstract boolean verschieben (Hour neu);  
};
```

Jede abstrakt deklarierte Methode muß in einer Unterklasse realisiert werden - sonst können keine Objekte der Unterklasse erzeugt werden!

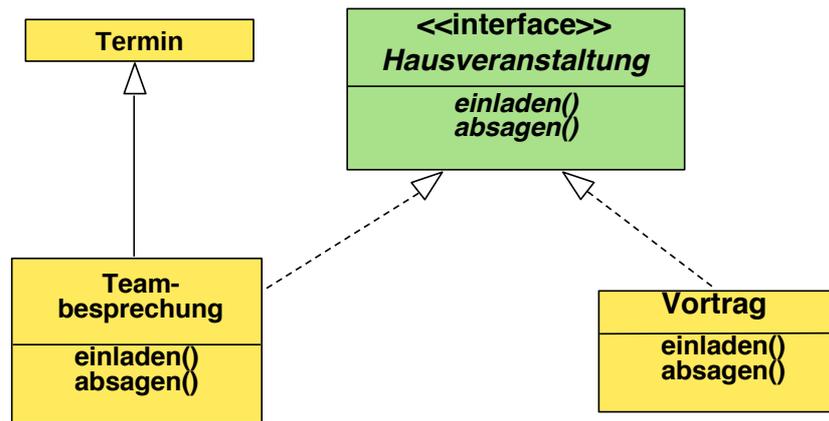
```
class Teambesprechung  
    extends Termin {  
    ...  
    boolean verschieben (Hour neu) {  
        boolean ok =  
            abstimmen(neu, dauer);  
        if (ok) {  
            beginn = neu;  
            raumFestlegen();  
        };  
        return ok;  
    };  
};
```

```
class Betriebsversammlung  
    extends Termin {  
    ...  
    boolean verschieben (Hour neu) {  
        beginn = neu;  
        raumFestlegen();  
        return ok;  
    };  
};
```

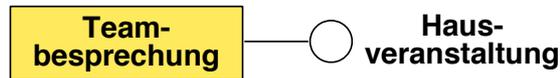
Fügt man eine abstrakte Methode hinzu, muss man auch die Klasse als abstrakt deklarieren.

Diese können dann in konkreten Unterklassen gefüllt werden.

Einfache Vererbung von Typen durch Schnittstellen



Hinweis: "Lutscher"-Notation (*lollipop*) für Schnittstellen „Klasse bietet Schnittstelle an“:



Das Abteil "angebotene Funktionen" kann in UML durch einen Lollipop benannt und abgekürzt werden.

Vergleich von Schnittstellen und abstrakte Klassen

Abstrakte Klasse

Enthält Attribute und Operationen

Kann Default-Verhalten festlegen

Wiederverwendung von Schnittstellen und Code, aber keine Instanzbildung

Default-Verhalten kann in Unterklassen überdefiniert werden

Java: Unterklasse kann nur von einer Klasse erben

Schnittstelle (voll abstrakt)

Enthält nur Operationen (und ggf. Konstante)

Kann kein Default-Verhalten festlegen

Redefinition unsinnig

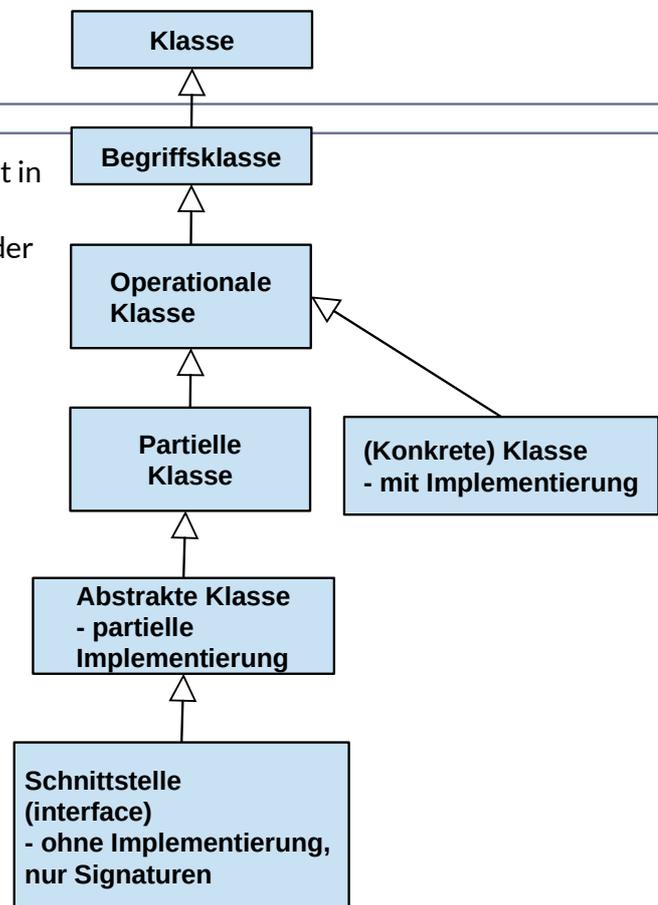
Java und UML: Eine Klasse kann mehrere Schnittstellen implementieren

Schnittstelle ist eine spezielle Sicht auf eine Klasse

Q2: Begriffshierarchie von Klassen (Erw.)

15 Softwaretechnologie (ST)

- ▶ *Operationale Klassen* werden unterteilt in Klassen mit, ohne, und mit Implementierung einer Untermenge der Operationen
- ▶ *Schnittstellen* und *Abstrakte Klassen* dienen dem Teilen von Typen und partiellem Klassen-Code



Wieder eine Taxonomie, diesmal von Klassen. Partielle Klassen geben vor, wie man Unterklassen bzw. Typen bildet.

14.2. Generische Klassen (Klassenschablonen, Template-Klassen, Parametrische Klassen) ... bieten eine weitere Art, mit Löchern zu programmieren, um Vorgaben zu machen

- ▶ Generische Klassen lassen den Typ von einigen Attributen und Referenzen offen ("generisch")
- ▶ Sie ermöglichen typisierte Wiederverwendung von Code (Ausfaktorisierung von Gemeinsamkeiten)
- ▶ Sie helfen, Nachbarn zu spezialisieren



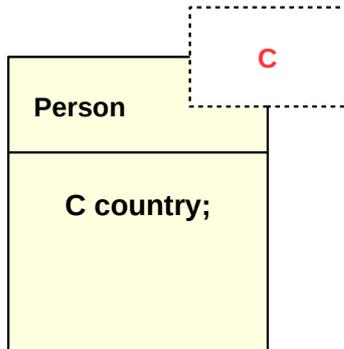
<https://www.uml-diagrams.org/template.html>

Generische Klassen bieten eine weitere Möglichkeit des "Programmierens mit Löchern". Während abstrakte Klassen einzelne Methoden "offen" lassen, lässt eine generische Klasse einen ganzen Typ offen, i.d.R. den Typ eines Attributs seiner Objekte.

Generische Klassen

Def.: Eine **generische (parametrische, Template-) Klasse** ist eine Klassenschablone, die mit einem oder mehreren Typparametern (für Attribute, Referenzen, Methodenparameter) versehen ist.

▶ In UML



▶ In Java

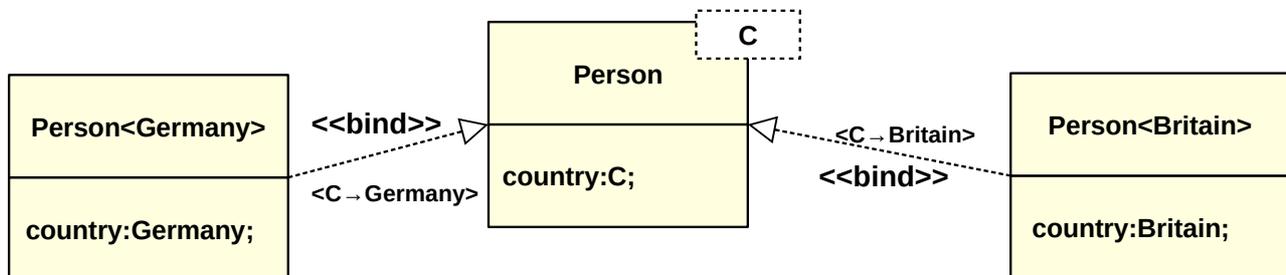
- Sprachregelung: "Person of C"

```
// Definition of a generic class
class Person<C> {
    C country;
}
```

```
/* Type definition using a generic type */
Person<Germany> egon;
Person<Britain> john;
```

Feinere statische Typüberprüfung für Attribute

- ▶ Zwei Attributtypen, die durch Parameterisierung aus einer generischen Klassenschablone entstanden sind, sind nicht miteinander kompatibel
- ▶ Der Übersetzer entdeckt den Fehler (**statische Typprüfung**)
- ▶ Die generische Klasse beschreibt das Gemeinsame (Generalisierung); der Parameter die Verschiedenheiten; die ausgeprägte Klasse die Spezialisierung



```
/* Type definition and initialization with object */
Person<Germany> egon = new Person<Germany>;
Person<Britain> john = new Person<Britain>;

/* Checks of assignments can use the improved typing */
john = egon;
```

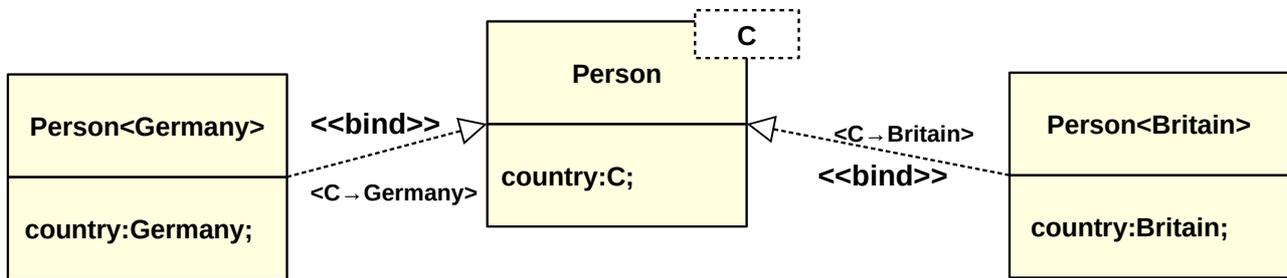
Die Parameterisierungsoperation heißt in UML “bind”:

- Ein Template wird an eine Instanz “gebunden” (“expandiert”), indem ein formaler Parameter des Templates an einen aktuellen Parameter gebunden wird

•

Einsatzzweck: Typen von Nachbarn vorgeben

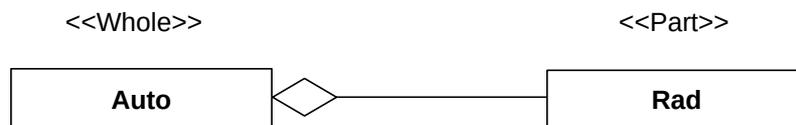
- ▶ **Def.:** Wenn ein formaler Typparameter einer generischen Klasse einen Nachbarn beschreibt, nennen wir ihn **Nachbartypschränke**



- ▶ **Bsp.:** C gibt für die generische Klasse den Typ des Landes country vor (Nachbar)
- ▶ C ist eine Nachbartypschränke

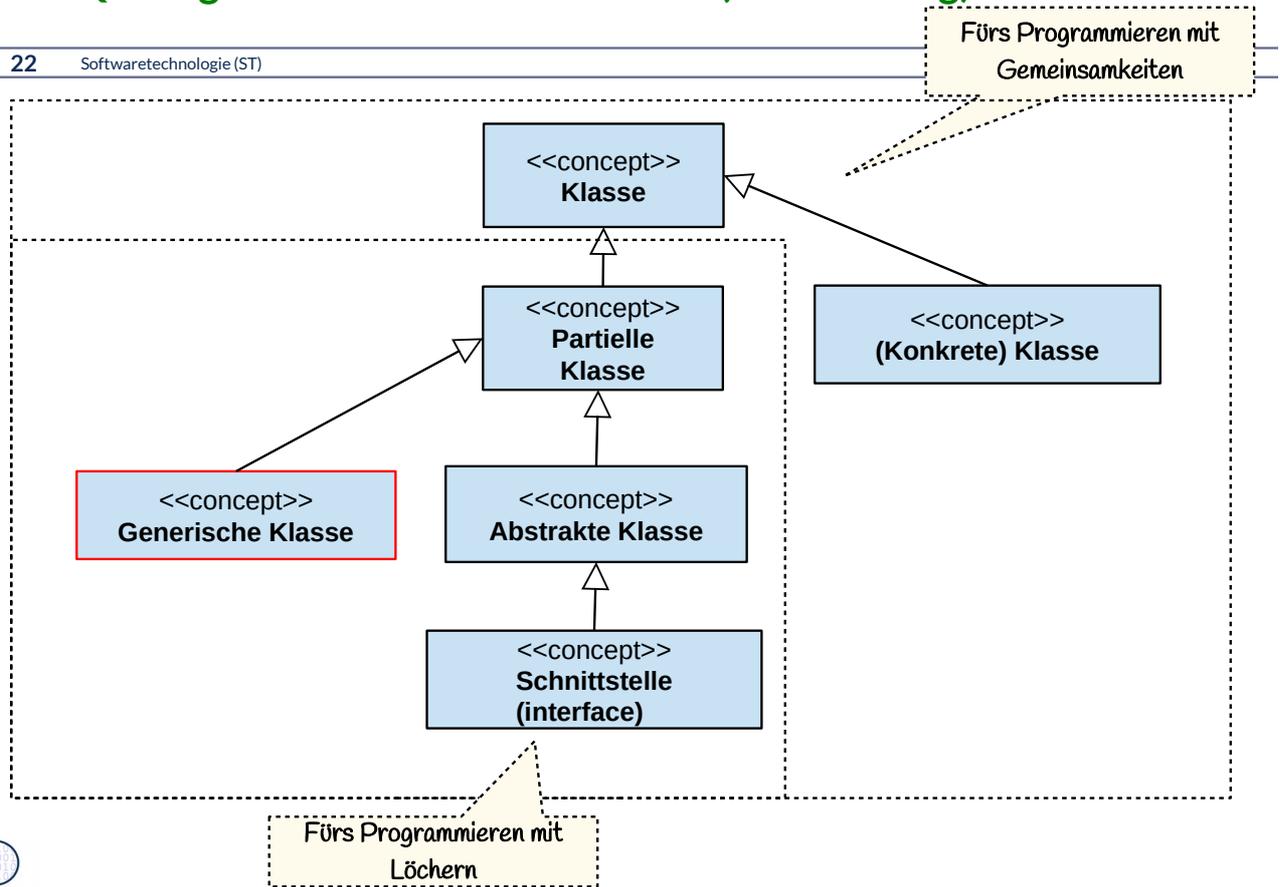
Einsatzzweck: Typen von Nachbarn vorgeben, Typsichere Aggregation (has-a)

- ▶ **Def.:** Wenn eine Assoziation den Namen „hat-ein“ oder „besteht-aus“ tragen könnte, handelt es sich um eine **Aggregation** zwischen einem *Aggregat*, dem *Ganzen*, und seinen *Teilen* (Ganzes/Teile-Relation, whole-part relationship).
 - Die auftretenden Aggregationen bilden auf den Objekten immer eine transitive, antisymmetrische Relation (einen gerichteten zyklensfreien Graphen, *dag*).
 - Ein Teil kann zu mehreren Ganzen gehören (*shared*), zu einem Ganzen (*owns-a*) und exklusiv zu einem Ganzen (*exclusively-owns-a*)



Lies: „Auto *hat ein* Rad“

Q2: Begriffshierarchie von Klassen (Erweiterung)



Jetzt können wir verstehen, welche Unterkonzepte der “Klasse” zum Programmieren mit Löchern vorgesehen sind.

- ▶ Schnittstellen als auch abstrakte Klassen erlauben es, Anwendungsprogrammierern Struktur vorzugeben
 - Sie definieren “Haken”, in die Unterklassen konkrete Implementierungen schieben
 - Schnittstellen sind vollständig abstrakte Klassen
- ▶ Generische Klassen ermöglichen typsichere Wiederverwendung von Code über Typ-Parameter → der Compiler meldet mehr Fehler

Warum ist das wichtig?

- ▶ **Bau von Frameworks (Rahmenwerken)** ist eines der Hauptprobleme des Software Engineering
 - Von Projekt zu Projekt
 - Von Produkt zu Produkt (Produktfamilien, Produktlinien)
- ▶ Abstrakte Klassen, Schnittstellen und generische Klassen können Code-Replikat und Code-Explosion weitgehend vermeiden und **gleichzeitig Vorgaben für Erweiterungen machen**
- ▶ Wiederverwendung **mit Frameworks** ist das Hauptmittel der Softwarefirmen, um **profitabel** arbeiten zu können

Verständnisfragen

- ▶ Geben Sie eine Begriffshierarchie des Klassenbegriffs an. Welche Klassenarten kennen Sie? Wie spezialisieren sie sich?
- ▶ Erklären Sie den Unterschied der “Löcher” in abstrakten Klassen und in generischen Klassen.
- ▶ Warum wird die Wiederverwendung von Software durch Frameworks vereinfacht? Wozu gibt der Framework-Konstrukteur Vorgaben für die Löcher vor?

